matCUDA operation guide

64-bits WINDOWS OS

version 1.0

Summary

1. INTRODUCTION

matCUDA is a free open-source C++ library that aims to facilitate the development of C++ codes that involves generalized vectors and matrix operations with high performance. An instance of the herein presented Array class represents an array (vector or matrix), whose needed memory is automatically allocated at the class instantiation. Mathematical operators are overloaded, providing intuitive ways of perform algebraic operations. Besides, there are some algebraic functions implemented. Most of the operations and functions are CUDA-implemented, in order to take advantage of the high performance provided by the use of GPUs onto computation.

The Array class is the most important class contained in matCUDA library and is responsible to interface between the user and the other classes, responsible for automatic allocation/deallocation of memory, data transfer to and from the GPU, GPU and CPU operations, etc. It has some mathematical operators overloaded, in a way that a multiplication of two instances of Array is performed through the operator "*", and its sum by "+", and most of its functionalities are implemented as public members.

In this text, the expression "desired Array" means the self-object, the Array itself, the one represented on the left side of the operator ".", the accessed with the private word "this". In the operator case, it is the Array to the left of the operator.

2. CONSTRUCTORS

There are three main ways of creating an object of Array. 1 – specifying its dimensions; 2 – copy constructor and 3 – operator "=" constructor. They are all shown in the Listing 1 below.

```
#include <matCUDA.h>
void example_01()
{
	size_t N = 1024;

	// creates float-vector with N elements
	Array<float> v1( N );

	// creates double-matrix with N x N elements
	   Array<float> m1( N, N );

	// creates std::complex<double>-hypermatrix
	   // with N x N x N x N elements
	   Array<ComplexDouble> m2( N, N, N, N );

	   // fills in v1 and m1 with random numbers
	   v1 = rand<float>( N );
	   m1 = rand<float>( N, N );

	   // creates v2 with same type, dimensions
	   // and element-values as m1*v1, which
	   // has the same dimensions as v1
	   Array<float> v2 = m1*v1;
}
```

Listing 1: Examples of ways of create Arrays, i. e., to instantiate the matCUDA class

Notice that the maximum dimensions allowed is 32; however, algebraic operations are defined for number of dimensions equal 2 (matrix) or 1 (vector).

3. MEMBER FUNCTIONS

Almost all the functions are templates, that's why the type is inside <>, as <double>, for example. The five types allowed are int, float, double, ComplexFloat (typedef of std::complex<float>) and ComplexDouble (typedef of std::complex<double>). Not all the operations are defined for every type, however it will be specified the types allowed for each function. The generic type is represented by the word "TElement" on the functions description.

There are two main groups of functions. First group presented represents member functions that perform algebraic operations on Arrays, such as matrices inversion, decomposition, etc. Second group represents member functions that evaluate information on the desired Array, as number of dimensions, length of dimensions, number of elements, etc.

On the members that manipulate Array

Below there is Listing 2, which contains the declaration of such functions inside class Array. Now a brief explanation of each function.

3.1. Array<TElement>   abs();

- Returns an Array whose elements are the elementwise absolute value of the elements of the desired Array.
- Defined for types ComplexFloat and ComplexDouble.
- Implemented on GPU.

3.2. Array<TElement>   abs2();

- Returns an Array whose elements are the elementwise absolute value squared of the elements of the desired Array.
- Defined for types ComplexFloat and ComplexDouble.
- Implemented on GPU.

3.3. Array<TElement>   acos();

- Returns an Array whose elements are the elementwise arc cosine (in radians) of the elements of the desired Array.
- Defined for types float and double.
- Implemented on CPU.

```
        // Member functions
        Array<TElement>     abs(); // gpu
        Array<TElement>     abs2(); // gpu
        Array<TElement>     acos();
        Array<TElement>     acosd();
        Array<TElement>     asin();
        Array<TElement>     asind();
        Array<TElement>     atan();
        Array<TElement>     atand();
        bool                check_close( Array<TElement> a );
        Array<TElement>     conj(); // gpu
        Array<TElement>     conjugate(); // gpu
        Array<TElement>     cos();
        Array<TElement>     cosd();
        TElement            determinant(); // gpu
        Array<TElement>     detrend(); // gpu + cpu
        Array<TElement>     diff();
        Array<TElement>     elementWiseDivide( Array<TElement> *A ); // gpu
        Array<TElement>     elementWiseMultiply( Array<TElement> *A );
        Array<TElement>     fft(); // gpu
        Array<TElement>     getColumn( const index_t col );
        Array<TElement>     hermitian(); // gpu
        Array<TElement>     invert(); // gpu
        Array<TElement>     ls( Array<TElement> *A ); // gpu
        Void                lu( Array<TElement> *L, Array<TElement> *U, Array<TElement> *P ); // gpu
        void                lu( Array<TElement> *L, Array<TElement> *U ); // gpu
        Array<TElement>     max(); // gpu
        Array<TElement>     max( Array<TElement> *idx ); // gpu
        Array<TElement>     min(); // gpu
        Array<TElement>     min( Array<TElement> *idx ); // gpu
        Array<TElement>     minor(const int row, const int column);
        TElement            norm(); // gpu
        void                print();
        void                qr( Array<TElement> *Q, Array<TElement> *R ); // gpu
        Array<TElement>     removeCol( const index_t col );
        Array<TElement>     removeRow( const index_t row );
        Array<TElement>     sin();
        Array<TElement>     sind();
        Array<TElement>     submatrix( const index_t rowBegin, const index_t rowEnd, const index_t
colBegin, const index_t colEnd );
        Array<TElement>     tan();
        Array<TElement>     tand();
        Array<TElement>     transpose(); // gpu
        void                write2file( std::string s = arrayname2str() );
```

Listing 2: member functions of Array

3.4. Array<TElement> acosd();

- Returns an Array whose elements are the elementwise arc cosine (in degrees) of the elements of the desired Array.
- Defined for types float and double.
- Implemented on CPU.

3.5. Array<TElement> asin();

- Returns an Array whose elements are the elementwise arc sine (in radians) of the elements of the desired Array.
- Defined for types float and double.

- Implemented on CPU.

3.6. Array<TElement> asind();

- Returns an Array whose elements are the elementwise arc sine (in degrees) of the elements of the desired Array.
- Defined for types float and double.
- Implemented on CPU.

3.7. Array<TElement> atan();

- Returns an Array whose elements are the elementwise arc tangent (in radians) of the elements of the desired Array.
- Defined for types float and double.
- Implemented on CPU.

3.8. Array<TElement> atand();

- Returns an Array whose elements are the elementwise arc tangent (in degrees) of the elements of the desired Array.
- Defined for types float and double.
- Implemented on CPU.

3.9. bool check_close(Array<TElement> a);

- Compares the input Array "a" and the desired Array elementwise. If all values are very close to each other, it returns "true", otherwise "false". It is meant to substitute (regular) operator "==" (equal), since "`check_close`" takes in account for rounding errors, and considers values that are very close to each other as "equal".
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

3.10. Array<TElement> conj();

- Returns an Array whose elements are the elementwise complex-conjugated values of the elements of the desired Array.
- Defined for types int, float, double, ComplexFloat and ComplexDouble. It returns the same Array for types int, float and double (once they are real and not complex Arrays).
- Implemented on GPU.

3.11. Array<TElement> conjugate();

- Same of `Array<TElement> conj();`

3.12. Array<TElement> cos();

- Returns an Array whose elements are the elementwise cosine of the elements of the desired Array, considered in radians.
- Defined for types float and double.
- Implemented on CPU.

3.13. Array<TElement> cosd();

- Returns an Array whose elements are the elementwise cosine of the elements of the desired Array, considered in degrees.
- Defined for types float and double.
- Implemented on CPU.

3.14. TElement determinant();

- Returns an element that corresponds to the determinant of the desired Array.
- Meant for square Arrays i.e. Arrays with two equal dimensions (square matrix).
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.

3.15. Array<TElement> detrend();

- Returns an Array that represents the removal of the linear trend present in the desired Array. It performs a least-square regression to fit a line to the Array elements, then removes this trend.
- Meant for vector Arrays i.e. Arrays with one dimension.
- Defined for types float, double, ComplexFloat and ComplexDouble, although tested just for float and double.
- Implemented on GPU.

3.16. Array<TElement> diff();

- Returns an Array that represents first forward differencing of the desired Array.
- Meant for vector Arrays i.e. Arrays with one dimension. If the input is a matrix (two dimensions), the operation will take place on each column of the desired Array independently.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

3.17. Array<TElement> elementWiseDivide(Array<TElement> *A);

- Returns an Array that represents the elementwise division of the desired Array by the Array represented by *A.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

3.18. Array<TElement> elementWiseMultiply(Array<TElement> *A);

- Returns an Array that represents the elementwise multiplication of the desired Array by the Array represented by *A.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.

3.19. Array<TElement> fft();

- Returns an Array that represents the fourier transform of each column of the desired Array.
- Defined for types ComplexFloat and ComplexDouble.
- Implemented on GPU.

3.20. Array<TElement> getColumn(const index_t col);

- Returns an Array that represents the column **col** of the desired Array.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

3.21. Array<TElement> hermitian();

- Returns an Array that represents the hermitian of the desired Array.
- Defined for types float, double, ComplexFloat and ComplexDouble. It returns the transposed Array for types int, float and double (once they are real and not complex Arrays). It returns the transposed complex-conjugated Array for ComplexFloat and ComplexDouble.
- Implemented on GPU.

3.22. Array<TElement> invert();

- Returns an Array that represents the inverse matrix of the desired Array.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Meant for square Arrays i.e. Arrays with two equal dimensions (square matrix).
- Implemented on GPU.

3.23. Array<TElement> ls(Array<TElement> *A);

- Returns an Array that represents the solution of a linear system of the kind $A.x = b$. It returns an Array that corresponds to the parameters x on the equation. The desired Array is the "data" Array, represented by b on the equation, and A is represented by *A.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.

3.24. void lu(Array<TElement> *L, Array<TElement> *U, Array<TElement> *P);

- Performs the LU decomposition of the desired Array, in a way that $P.A = L.U$, where P is the pivoting (*P), L is the lower-triangular (*L) and U is the upper-triangular (*U) matrices.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.
- TOO SLOW. No recommended if pivoting matrix not necessary.

3.25. void lu(Array<TElement> *L, Array<TElement> *U);

- Performs the LU decomposition of the desired Array, in a way that $P.A = L.U$, where P is the pivoting (*P), L is the lower-triangular (*L) and U is the upper-triangular (*U) matrices.

- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.
- Recommended if pivoting matrix not necessary.

3.26. Array<TElement> max();

- Returns an Array that represents the maximum element of each column of the desired Array.
- Defined for types float and double.
- Implemented on GPU.

3.27. Array<TElement> max(Array<TElement> *idx);

- Returns an Array that represents the maximum element of each column of the desired Array and stores in *idx the index of each maximum value.
- Defined for types float and double.
- Implemented on GPU.

3.28. Array<TElement> min();

- Returns an Array that represents the minimum element of each column of the desired Array.
- Defined for types float and double.
- Implemented on GPU.

3.29. Array<TElement> min(Array<TElement> *idx);

- Returns an Array that represents the minimum element of each column of the desired Array and stores in *idx the index of each minimum value.
- Defined for types float and double.
- Implemented on GPU.

3.30. Array<TElement> minor(const index_t row ,const index_t column);

- Returns an Array that represents the minor matrix of the desired Array, with row **row** and column **column** excluded.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Meant for matrices Arrays i.e. Arrays with two dimensions.
- Implemented on CPU.

3.31. TElement norm();

- Returns an element that represents the norm of the desired Array.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Meant for vector Arrays i.e. Arrays with one dimension.
- Implemented on GPU.

3.32. void print();

- Print the desired Array on the screen.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

3.33. void qr(Array<TElement> *Q, Array<TElement> *R);

- Performs the QR decomposition of the desired Array, in a way that $A = Q.R$, where Q is an orthogonal matrix (*Q) and R is an upper-triangular (*R) matrix.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.

3.34. Array<TElement> removeCol(const index_t col);

- Returns an Array that represents the minor matrix of the desired Array, in which column **col** is excluded.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Meant for matrices Arrays i.e. Arrays with two dimensions.
- Implemented on CPU.

3.35. Array<TElement> removeRow(const index_t row);

- Returns an Array that represents the minor matrix of the desired Array, in which row **row** is excluded.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Meant for matrices Arrays i.e. Arrays with two dimensions.
- Implemented on CPU.

3.36. Array<TElement> sin();

- Returns an Array whose elements are the elementwise sine of the elements of the desired Array, considered in radians.
- Defined for types float and double.
- Implemented on CPU.

3.37. Array<TElement> sind();

- Returns an Array whose elements are the elementwise sine of the elements of the desired Array, considered in degrees.
- Defined for types float and double.
- Implemented on CPU.

3.38. Array<TElement> submatrix(const index_t rowBegin, const index_t rowEnd, const index_t colBegin, const index_t colEnd);

- Returns an Array that represents the minor matrix of the desired Array, which includes rows ranging from **rowBegin** to **rowEnd** and columns ranging from **colBegin** to **colEnd**.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Meant for matrices Arrays i.e. Arrays with two dimensions.
- Implemented on CPU.

3.39. Array<TElement> tan();

- Returns an Array whose elements are the elementwise tangent of the elements of the desired Array, considered in radians.

- Defined for types float and double.
- Implemented on CPU.

3.40. Array<TElement> tand();

- Returns an Array whose elements are the elementwise tangent of the elements of the desired Array, considered in degrees.
- Defined for types float and double.
- Implemented on CPU.

3.41. Array<TElement> transpose();

- Returns an Array that represents the transpose of the desired Array.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.

3.42. void write2file(std::string s = arrayname2str());

- Writes the desired Array to file defined in string **s**. If not provided as argument, it will be saved on a file with the name of the desired Array. It writes with the Array disposal of elements.
- Meant for matrices Arrays i.e. Arrays with two dimensions.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

On the members that extract information about Array

Below there is Listing 3, which contains the declaration of such functions inside class Array. Now a brief explanation of each function.

```
        // info from base classes
        TElement*                       data();
        size_t                          getDim( index_t dim );
        size_t                          getNDim();
        size_t                          getNElements();
```

Listing 3: members function of Array

3.43. TElement* data();

- Returns a pointer to where Array data is allocated.

3.44. size_t getDim(index_t dim);

- Returns the length of the dimension dim.

3.45. size_t getNDim();

- Returns the amount of dimensions.

3.46. size_t getNElements();

- Returns the amount of elementes.

4. OPERATORS

Below there is Listing 4, which contains the declaration of such operators inside class Array. Now a brief explanation of each one.

```
// operators
bool operator                  == (Array<TElement> a);
bool operator                  != (Array<TElement> a);
Array<TElement>& operator   = (Array<TElement> &a);
Array<TElement>& operator   = (TElement a);
Array<TElement> operator    + (Array<TElement>  &a); // gpu
Array<TElement> operator    + (TElement a);
Array<TElement> operator    - (Array<TElement> &a); // gpu
Array<TElement> operator    - (TElement a);
Array<TElement> operator    * (Array<TElement> &a); // gpu
Array<TElement> operator    * (TElement a);
Array<TElement> operator    / (Array<TElement> &a); // gpu
Array<TElement> operator    / (TElement a);
Array<TElement>& operator  += (Array<TElement> &a); // gpu
Array<TElement>& operator  -= (Array<TElement> &a); // gpu
Array<TElement>& operator  *= (Array<TElement> &a); // gpu
```

Listing 4: operators overloaded for class Array

4.1. bool operator == (Array<TElement> a);

- uses bool check_close(Array<TElement> a);
- Compares both Arrays elementwise. If at least one is different, returns false.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Meant for Arrays with exactly the same dimensions.
- Implemented on CPU.

4.2. bool operator =! (Array<TElement> a);

- uses bool check_close(Array<TElement> a);
- Compares both Arrays elementwise. If at least one is different, returns true.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Meant for Arrays with exactly the same dimensions.
- Implemented on CPU.

4.3. Array<TElement>& operator = (Array<TElement>& a);

- Returns an Array whose elements are equal the elements of Array **a**.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

4.4. Array<TElement>& operator = (TElement a);

- Returns an Array whose elements are all equal the value of **a**..

- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

4.5. Array<TElement> operator + (Array<TElement>& a);

- Returns an Array whose elements are the elementwise sum of the desired Array with Array **a**.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Meant for Arrays with exactly the same dimensions.
- Implemented on GPU.

4.6. Array<TElement> operator + (TElement a);

- Returns an Array whose elements are the elementwise sum of the desired Array with element **a**.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

4.7. Array<TElement> operator - (Array<TElement>& a);

- Returns an Array whose elements are the elementwise subtraction of the desired Array with Array **a**.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Meant for Arrays with exactly the same dimensions.
- Implemented on GPU.

4.8. Array<TElement> operator - (TElement a);

- Returns an Array whose elements are the elementwise subtraction of the desired Array with element **a**.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

4.9. Array<TElement> operator * (Array<TElement>& a);

- Returns an Array that is the algebraic multiplication of the desired Array with Array **a**.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- This operator cover all the multiplication cases e.g. *"matrix * matrix = matrix"* or *"matrix * vector = vector"*. So, beware!
- Implemented on GPU.

4.10. Array<TElement> operator * (TElement a);

- Returns an Array whose elements are the elementwise multiplication of the desired Array with element **a**.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

4.11. Array<TElement> operator / (Array<TElement>& a);

- Returns an Array that is the algebraic multiplication of the desired Array with the inverse of Array **a**.

- Defined for types float, double, ComplexFloat and ComplexDouble.
- Meant for Arrays with exactly the same dimensions.
- Implemented on GPU.

4.12. Array<TElement> operator / (TElement a);

- Returns an Array whose elements are the elementwise division of the desired Array with element **a**.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

4.13. Array<TElement> operator += (Array<TElement>& a);

- Returns an Array whose elements are the elementwise sum of the desired Array with Array **a**, and stores the result in the desired Array.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Meant for Arrays with exactly the same dimensions.
- Implemented on GPU.

4.14. Array<TElement> operator -= (Array<TElement>& a);

- Returns an Array whose elements are the elementwise subtraction of the desired Array with Array **a**, and stores the result in the desired Array.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Meant for Arrays with exactly the same dimensions.
- Implemented on GPU.

4.15. Array<TElement> operator *= (Array<TElement>& a);

- Returns an Array that is the algebraic multiplication of the desired Array with Array **a**, and stores the result in the desired Array.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Meant for *"matrix * matrix = matrix"* and *"vector.transpose() * matrix = vector.transpose()"*.
- Implemented on GPU.

5. GLOBAL FUNCTIONS

Below there is Listing 5, which contains the declaration of such functions inside class Array. Now a brief explanation of each function.

5.1. Array<TElement> eye(index_t N);

- Returns a square Array (two equal dimensions) that is equal to the identity matrix of size **N** x **N**.
- Defined for types int, float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.

```
template <typename TElement>
Array<TElement>                    eye( index_t N );

template <typename TElement>
Array<std::complex<TElement>>      fft( Array<TElement> *in );

template <typename TElement>
Array<TElement>                    rand(index_t u);

template <typename TElement>
Array<TElement>                    rand(index_t u1, index_t u2);

template <typename TElement>
Array<TElement>                    read_file_vector( std::string s );

template <typename TElement>
Array<TElement>                    read_file_matrix( std::string s );

// measure time and print
void                               tic();
void                               toc();
long double                        toc( long double in );
```

Listing 5: global functions

5.2. Array<std::complex<TElement>> fft(Array< TElement > *in);

- Returns a complex Array that represents the symmetric Fourier transform of each column of the real input Array **in**. If **in** has N rows, the returned Array has $N/2 + 1$ rows, and the same amount of columns.
- Defined for types float and double.
- Implemented on GPU.

5.3. Array< TElement> rand(index_t u);

- Returns a vector Array filled with **u** random numbers.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.

5.4. Array< TElement> rand(index_t u1, index_t u2);

- Returns a matrix Array with dimensions **u1** x **u2** filled with random numbers.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on GPU.

5.5. Array< TElement> read_file_vector(std::string s);

- Returns a vector Array with elements contained in file described by string **s**. The first element of the file must be the dimension of the vector.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

5.6. Array< TElement> read_file_matrix(std::string s);

- Returns a vector Array with elements contained in file described by string **s**. The first line of the file must contains the dimensions of the matrix.
- Defined for types float, double, ComplexFloat and ComplexDouble.
- Implemented on CPU.

5.7. void tic();

- Start the timer.

5.8. void toc();

- Stop the timer and print the elapsed time between a “tic()” and a “toc()” on the screen.

5.9. long double toc();

- Stop the timer and returns the elapsed time between a “tic()” and a “toc()”.